Czym jest MySQL?
MySQL jest szybkim, wielowątkowym serwerem baz danych obsługującym język zapytań SQL. Pracuje z wieloma użytkownikami i doskonale nadaja się do wykorzystania razem z PHP jako darmowa platforma aplikacji internetowych.
W MySQL-u możesz między innymi tworzyć nowe bazy danych, a w nich tabele, dodawać nowe rekordy, edytować je lub usuwać. Masz do dyspozycji możliwość zarządzania użytkownikami, w tym również dokładnego określania praw dostępu do określonego pola w bazie danych.
W danym momencie z bazy danych może korzystać nieograniczona liczba osób; zależy to jedynie od zasobów fizycznych serwera.
Wszystkie przykłady obsługi MySQL-a w tym kursie będą opierać się na lokalnej bazie danych, zainstalowanej pod systemem Windows.
Logowanie
Pierwszą rzeczą, jaką chciałbyś zrobić jest pewnie 'zobaczenie' MySQL-a. Musisz się do niego najpierw zalogować. W okienku DOS-a przejdź do katalogu z binariami MySQL-a. Domyślnie jest to katalog c:\mysql\bin

```
c:\windows> cd ..
c:\> cd mysql\bin
c:\mysql\bin>
```

Następnie wpisz 'mysql' z parametrami -u (użytkownik) oraz jeśli wymagane jest hasło to również -p.

```
c:\mysql\bin> mysql -u root -p
Enter password: *******
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 24 to server version 3.23.22-beta-debug
Type 'help' for help.
mysql>
```

MySQL wypisuje zarówno informacje na temat serwera bazy danych, jak i numer identyfikacyjny połączenia z bazą danych. Mamy również do dyspozycji znak zachęty MySQL-a (mysql>).
Jeśli instalowałeś pakiet PHP Triad, to MySQL nie ma jeszcze ustawionego hasła dla użytkownika root i nie ma potrzeby jego podawania.
Składnia wywołania mysql przedstawia się następująco:
mysql [-h adres_hosta] [-u nazwa_użytkownika] [-p twoje_haslo]
Adres hosta to adres komputera, na którym znajduje się serwer bazy danych. Domyślnie jest to adres komputera lokalnego (localhost). MySQL domyślnie loguje właśnie do tego hosta.
Wyświetlenie istniejących baz danych
Z pewnością chciałbyś wiedzieć, jakie bazy danych są dostępne, zanim zaczniesz tworzyć nowe, własne bazy. Aby wyświetlić listę dostępnych baz danych posłużmy się poleceniem show databases. Zaloguj się do MySQL-a (jak to zrobić) i wydaj polecenie show databases;:

```
mysql> show databases;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+

2 rows in set (0.00 sec)
```

Jak już pewnie zauważyłeś, polecenie show databases zostało zakończone znakiem średnika (podobnie jak w PHP). SQL wymaga kończenia zapytań średnikiem, co pozwala na zapisywanie jednej instrukcji w wielu liniach. Spójrzmy na przykład:

```
mysql> show databases
   -> ;
+----------+
| Database |
+----------+
| mysql    |
| test     |
+----------+
2 rows in set (0.00 sec)

mysql>
```

Zwróć uwagę, że nie został podany średnik po poleceniu i MySQL czeka na kolejne dane. Dopiero kiedy zakończysz linię średnikiem, polecenie zostanie wykonane.

Jak widać, w systemie mamy utworzone dwie bazy danych, mysql i test. Pierwsza to baza o specjalnym znaczeniu; to w niej MySQL przechowuje wszystkie dane na temat użytkowników i ich uprawnień. Druga to specjajnie utworzona, pusta baza danych do testów.

Tworzenie nowej bazy danych

Aby rozpocząć naukę MySQL-a, utwórzmy najpierw naszą pierwszą, przeznaczoną do testów bazę danych. Tworzenie zarówno tabel, jak i baz odbywa się za pomocą polecenia create z odpowiednimi argumentami (w tym przypadku database i nazwą bazy, którą chcemy utworzyć):

```
mysql> create database nasza_baza;
Query OK, 1 row affected (0.06 sec)

mysql>
```

Właśnie utworzyliśmy nową bazę danych o nazwie nasza_baza. MySQL poinformował nas o powodzeniu i czasie wykonania polecenia. Spójrzmy zatem na listę dostępnych baz danych:

```
mysql> show databases;
+-------------+
| Database    |
+-------------+
| mysql       |
| test        |
| nasza_baza |
+-------------+
3 rows in set (0.00 sec)

mysql>
```

Jak widać pojawiła się tam nowa pozycja, nasza_baza.
Usuwanie bazy danych
Ponieważ utworzyliśmy własną bazę danych do testów, nie jest już nam potrzebna baza test. Usuńmy ją za pomocą polecenia drop z odpowiednimi argumentami (w tym przypadku database i nazwą bazy, którą chcemy usunąć):

```
mysql> drop database test;
Query ok, 0 rows affected (0.00 sec)

mysql>
```

Sprawdźmy, jakie bazy są teraz dostępne:

```
mysql> show databases;
+------------+
| Database   |
+------------+
| mysql      |
| nasza_baza |
+------------+
2 rows in set (0.00 sec)

mysql>
```

Jak widać pozycja test nie jest wyświeltana, ponieważ została usunięta wraz z danymi, które mogły się tam znajdować.

Wybieranie bazy danych

Aby możliwe było zarządzanie tabelami (tworzenie, edycja bądź usuwanie) musimy najpierw 'powiedzieć' MySQL-owi, nad którą bazą danych mamy zamiar pracować. Można to zrobić na dwa sposoby:

podając ścieżkę do tabeli w notacji <nazwa_bazy>.<tazwa_tabeli>,

wywołując komendę use <nazwa_bazy>. Wtedy nie musimy już podawać przed nazwą tabeli nazwy bazy, ponieważ domyślnie to w niej MySQL szuka podanej tabeli.

Spójrzmy na przykład z zastosowaniem komenty use:

```
mysql> use nasza_baza
Database changed

mysql>
```

Wyświetlanie istniejących tabel

Kiedy baza została już wybrana (w tym przypadku nasza_baza), możemy przystąpić do wyświetlania tabel. Do tego celu służy poznana wcześniej komenda show z argumentem tables:

```
mysql> show tables
Empty set (0.00 sec)

mysql>
```

Zwrócony komunikat świadczy o tym, że baza nasza_baza jest pusta. Teraz spróbujemy wyświetlić tabelę w bazie mysql; wcześniej jednak skorzystamy z komendy use, aby zmienić bazę, nad którą chcemy pracować:

```
mysql> use mysql
Database changed

mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| host            |
| tables_priv     |
| user            |
+-----------------+
5 rows in set (0.00 sec)

mysql>
```

Tworzenie tabeli

Skoro mamy już wcześniej stworzoną bazę danych nasza_baza, warto byłoby utworzyć w niej nową tabelę. Do tego celu służy poznana już wcześniej komenda create z argumentem table <nazwa_tabeli> oraz definicją listy pól. Wybrane typy pól są przedstawione poniżej:

CHAR(M)
Pole znakowe. Przechowuje teksty o ustalonej z góry długości. Ograniczone do 255 znaków.

VARCHAR(M)
Pole znakowe. Przechowuje taką długość tekstu, jaka jest używana.

INT[(M)] [UNSIGNED]
Pole liczb całkowitych. Przechowuje liczby z zakresu od -2147483648 do 2147483647 (z parametrem UNSIGNED - od 0 do 4294967295).

DATE
Pole daty przechowuje daty z zakresu od '1000-01-01' do '9999-12-31'.

BLOB/TEXT
Pole tekstowe. Przechowuje dłuższe, wielowierszowe teksty do 65535 znaków.

Spróbujmy teraz utworzyć tabelę do przechowywania adresów przyjaciół:

```
mysql> use nasza_baza
Database changed

mysql> CREATE TABLE adresy (
    -> id INT NOT NULL auto_increment,
    -> imie VARCHAR(15) DEFAULT '<podaj imie>',
    -> nazwisko VARCHAR(25) DEFAULT '<podaj nazwisko>',
    -> adres BLOB,
    -> PRIMARY KEY (id)
    -> );
Query OK, 0 rows affected (0.06 sec)

mysql>
```

Właśnie utworzyliśmy tabelę adresy w bazie nasza_baza. Nowa tabela zawiera cztery pola. Pole id pełni funkcję klucza podstawowego (PRIMARY KEY), który jednoznacznie identyfikuje dany rekord. Jest ono typu auto_increment, co oznacza, że MySQL automatycznie będzie

wstawiał do niego wartość o jeden większą, a po usunięciu któregoś z rekordów wartości nie przesuną się 'w górę'; zawsze będą takiem same. Poza tym pole id nie może być puste (NOT NULL). Pola imie oraz nazwisko mają domyślnie wstawianą wartość, jeżeli nie została podana (DEFAULT).

Jeżeli chcesz obejrzeć strukturę tabeli użyj polecenia describe <nazwa tabeli>:

```
mysql> describe adresy;
+----------+-------------+------+-----+----------------+------------+
| Field    | Type        | Null | Key | Extra          | Privileges |
+----------+-------------+------+-----+----------------+------------+
| id       | int(11)     |      | PRI | auto_increment | zobacz *   |
| imie     | varchar(15) | YES  |     |                | zobacz *   |
| nazwisko | varchar(25) | YES  |     |                | zobacz *   |
| adres    | blob        | YES  |     |                | zobacz *   |
+----------+-------------+------+-----+----------------+------------+
4 rows in set (0.00 sec)

mysql>
```

* = select,insert,update,references
(użyte, aby zapis tabeli był bardziej czytelny)

MySQL wylistował nam wszystkie cztery pola, wyświeltił ich typy, klucze i wartości domyślne. Za pomocą tego polecenia możesz szybko zorientować się w strukturze każdej tabeli.

Możesz użyć również synonimicznej funkcji show columns from <nazwa_tabeli> dla wylistowania pól tabeli.

Sprawdźmy zatem co zwróci teraz polecenie show tables:

```
mysql> show tables;
+----------------------+
| Tables_in_nasza_baza |
+----------------------+
| adresy               |
+----------------------+
1 row in set (0.05 sec)

mysql>
```

Edycja tabeli

Po utworzeniu tabeli zawsze możesz jej strukturę poddać edycji. W naszej tabeli przydałoby się jeszcze jedno pole przechowujące adres e-mail przyjaciela. Do edycji tabel służy polecenie alter z argumentem table <nazwa_tabeli>:

```
mysql> ALTER TABLE adresy ADD adres_email VARCHAR(60);

Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Nowe pole o nazwie adres_email zostało już dodane, spójrzmy zatem na wynik naszej pracy:

```
mysql> describe adresy;
+-------------+-------------+------+-----+----------------+------------+
| Field       | Type        | Null | Key | Extra          | Privileges |
+-------------+-------------+------+-----+----------------+------------+
| id          | int(11)     |      | PRI | auto_increment | zobacz *   |
| imie        | varchar(15) | YES  |     |                | zobacz *   |
| nazwisko    | varchar(25) | YES  |     |                | zobacz *   |
| adres       | blob        | YES  |     |                | zobacz *   |
| adres_email | varchar(60) | YES  |     |                | zobacz *   |
+-------------+-------------+------+-----+----------------+------------+
5 rows in set (0.00 sec)

mysql>
```

* = select,insert,update,references
(użyte, aby zapis tabeli był bardziej czytelny)

Jak widać, nowe pole występuje już na liście, ale jego nazwa nie jest zbyt udana. Zmieńmy ją na email:

```
mysql> ALTER TABLE adresy CHANGE adres_email email VARCHAR(60);
Query OK, 0 rows affected (0.11 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Przy zmianie nazwy pola należy podać jego nową definicję (czyli typ pola). Spróbujmy zmienić maksymalną długość pola email ze 60 znaków na 80:

```
mysql> ALTER TABLE adresy MODIFY email VARCHAR(80);
Query OK, 0 rows affected (0.05 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Jak widać z powyższego przykładu, przy zmianie jedynie właściwości pola posługujemy się komendą MODIFY, a nie CHANGE, jak w poprzednim przykładzie, gdzie zmienialiśmy nazwę pola (kolumny).

Sprawdźmy jeszcze, jak wygląda struktura naszej tabeli po tych zmianach:

```
mysql> describe adresy;
+----------+-------------+------+-----+----------------+------------+
| Field    | Type        | Null | Key | Extra          | Privileges |
+----------+-------------+------+-----+----------------+------------+
| id       | int(11)     |      | PRI | auto_increment | zobacz *   |
| imie     | varchar(15) | YES  |     |                | zobacz *   |
| nazwisko | varchar(25) | YES  |     |                | zobacz *   |
| adres    | blob        | YES  |     |                | zobacz *   |
```

```
| email    | varchar(80) | YES  |     |                | zobacz *   |
+----------+-------------+------+-----+----------------+------------+
5 rows in set (0.00 sec)

mysql>
```

```
* = select,insert,update,references
(użyte, aby zapis tabeli był bardziej czytelny)
```

Spróbujmy jeszcze dokonać dwu modyfikacji w strukturze naszej tabeli. Najpierw usuńmy pole nazwisko:

```
mysql> ALTER TABLE adresy DROP COLUMN nazwisko;
Query OK, 0 rows affected (0.00 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

A następnie dodajmy pole telefon, które ma występować bezpośrednio po polu adres:

```
mysql> ALTER TABLE adresy ADD telefon VARCHAR(15) AFTER adres;
Query OK, 0 rows affected (0.06 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql>
```

Aby dodać nowe pole po innym, używa się komendy AFTER. Aby natomiast dodać pole na początku tabeli, trzeba użyć komendy FIRST.

Spójrzmy jak teraz wygląda struktura naszej tabeli:

```
mysql> describe adresy;
+---------+-------------+------+-----+----------------+------------+
| Field   | Type        | Null | Key | Extra          | Privileges |
+---------+-------------+------+-----+----------------+------------+
| id      | int(11)     |      | PRI | auto_increment | zobacz *   |
| imie    | varchar(15) | YES  |     |                | zobacz *   |
| adres   | blob        | YES  |     |                | zobacz *   |
| telefon | varchar(15) | YES  |     |                | zobacz *   |
| email   | varchar(80) | YES  |     |                | zobacz *   |
+---------+-------------+------+-----+----------------+------------+
5 rows in set (0.00 sec)

mysql>
```

```
* = select,insert,update,references
(użyte, aby zapis tabeli był bardziej czytelny)
```

I na koniec zmieńmy nazwę naszej tabeli z adresy na ksiazka_adresowa:

```
mysql> ALTER TABLE adresy RENAME ksiazka_adresowa;
Query OK, 0 rows affected (0.05 sec)

mysql>
```

Wyświetlmy teraz listę tabel w bazie nasza_baza:

```
mysql> show tables;
+----------------------+
| tables_in_nasza_baza |
+----------------------+
| ksiazka_adresowa     |
+----------------------+
1 row in set (0.00 sec)

mysql>
```

Usuwanie tabeli

Przećwiczmy teraz usuwanie tabel, ale nie na tabeli ksiazka_adresowa, ponieważ przyda się nam ona później. Utwórzmy nową tabelę do_usuniecia z jednym polem o nazwie test:

```
mysql> CREATE TABLE do_usuniecia (
    -> test CHAR(10)
    -> );
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Wyświetlmy listę tabel dla bazy nasza_baza:

```
mysql> show tables;
+----------------------+
| tables_in_nasza_baza |
+----------------------+
| do_usuniecia         |
| ksiazka_adresowa     |
+----------------------+
2 rows in set (0.00 sec)

mysql>
```

Usuńmy teraz tabelę do_usuniecia za pomocą poznanej już komendy DROP z parametrem table <nazwa_tabeli>:

```
mysql> DROP TABLE do_usuniecia;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Sprawdźmy listę tabel w naszej bazie:

```
mysql> show tables;
+----------------------+
| tables_in_nasza_baza |
+----------------------+
| ksiazka_adresowa     |
+----------------------+
1 row in set (0.00 sec)

mysql>
```

Dodawanie rekordów

Do dodawanie rekordów używa się polecenia INSERT. Spójrzmy na najprostrzy przykład dodawania nowego rekordu do naszej książki adresowej w testowanej bazie danych:

```
mysql> INSERT INTO nasza_baza.ksiazka_adresowa VALUES(0, 'Jan', 'ul. Wrocławska 5', '888-55-41', 'jan@email.pl');
Query OK, 1 row affected (0.00 sec)

mysql>
```

Jak już pewnie zauważyłeś, polecenie dodawania nowego rekordu składa się z następujących części:

INSERT INTO <nazwa_bazy>.<nazwa_tabeli> lub tylko <nazwa_tabeli>, jeżeli zostało wcześniej wywołane zostało polecenie use <nazwa_bazy>,

VALUES().

W nawiasach okrągłych po słowie kluczowym VALUES wypisujemy listę wartości dla pól rekordu, który zamierzamy podać. Jakie i jakiego typu są te pola sprawdzimy znanym już poleceniem:

```
mysql> describe ksiazka_adresowa;
+---------+-------------+------+-----+--------------+----------------+
| Field   | Type        | Null | Key | Default      | Extra          |
+---------+-------------+------+-----+--------------+----------------|
| id      | int(11)     |      | PRI | NULL         | auto_increment |
| imie    | varchar(15) | YES  |     | <podaj_imie> |                |
| adres   | blob        | YES  |     | NULL         |                |
| telefon | varchar(15) | YES  |     | NULL         |                |
| email   | varchar(80) | YES  |     | NULL         |                |
+---------+-------------+------+-----+--------------+----------------+
5 rows in set (0.00 sec)

mysql>
```

tabela została skrócona dla zwiększenia czytelności kodu

Ze struktury tabeli wynika, że jeżeli nie podamy imienia, zostanie zapisana wartość domyślna '<podaj_imie>'. Ale co zrobić, żeby nie podać wartości dla danego pola? Możemy oczywiście postąpić tak (wtedy jednak wartość domyślna nie zostanie wstawiona):

```
mysql> INSERT INTO ksiazka_adresowa VALUES(0, '', 'ul Krakowska 16', '777-33-84', 'zebra@zoo.pl');
Query OK, 1 row affected (0.06 sec)

mysql>
```

Jest jednak lepszy sposób - wpisanie listy pól, dla których zamierzamy podać wartości, w nawaisach okrągłych występujących po nazwie tabeli:

```
mysql> INSERT INTO ksiazka_adresowa(id, adres, telefon) VALUES(0, 'ul. Warszawska 15', '458-34-75');
Query OK, 1 row affected (0.06 sec)

mysql>
```

Zauważyłeś pewnie, że zamiast wpisywać kolejne numery id, podajemy za każdym razem zero, a przecież taki sam numer id we wszystkich rekordach nic nam nie da. Zauważ jednak, że pole id jest typu auto_increment i nie ma znaczenia co do niego wpisujemy, i tak będzie miało odpowiednio inkrementowaną wartość. Dlatego lepiej byłoby dodawać rekordy bez podawania pola id:

```
mysql> INSERT INTO ksiazka_adresowa(telefon, email) VALUES('471-84-36', 'my@email.pl');
Query OK, 1 row affected (0.06 sec)

mysql>
```

Spójrzmy co zwróci MySQL, gdy nie podamy wszystkich wartości, dla których zgłosiliśmy pola:

```
mysql> INSERT INTO ksiazka_adresowa(telefon, adres) VALUES('433-22-88');
ERROR 1136: Column count doesn't match value count at row 1

mysql>
```

Zwrócony komunikat świadczy o niezgodności listy pól z wartościami. Rekord nie został dodany do naszej tabeli.

Wybieranie rekordów

Chciałbyś pewnie wiedzieć, jak wygląda po tych operacjach nasza tabela ksiazka_adresowa. Jest na to sposób: polecenie SELECT, które służy do wybierania w rozmaity sposób rekordów z bazy danych. Podstawowa składnia tego polecenia wygląda następująco:

SELECT <lista_pól> FROM <nazwa_bazy>.<nazwa_tabeli>;

lub

SELECT <lista_pól> FROM <nazwa_tabeli>;

jeżeli wcześniej zostało użyte polecenie use <nazwa_bazy>. Spójrzmy zatem, co zwróci polecenie SELECT użyte w stosunku do naszej tabeli:

```
mysql> SELECT * FROM ksiazka_adresowa;
+----+--------------+------------------+-----------+--------------+
| id | imie         | adres            | telefon   | email        |
+----+--------------+------------------+-----------+--------------+
|  1 | Jan          | ul. Wrocławska 5  | 888-55-41 | jan@email.pl |
```

```
|  2 |              | ul Krakowska 16   | 777-33-84 | zebra@zoo.pl |
|  3 | <podaj imie> | ul. Warszawska 15 | 468-34-75 | NULL         |
|  4 | <podaj imie> | NULL              | 471-84-36 | my@email.pl  |
+----+--------------+-------------------+-----------+--------------+
4 rows in set (0.00 sec)

mysql>
```

Jak widać wszystko, do dodawaliśmy, jest zapisane w naszej bazie danych. Znak * użyty zamiast listy pól oznacza, że chcemy wybrać wszystkie dostępne pola tabeli.

Spróbujmy teraz napisać takie polecenie SELECT, które poda nam tylko numery id i imiona:

```
mysql> SELECT id, imie FROM ksiazka_adresowa;
+----+--------------+
| id | imie         |
+----+--------------+
|  1 | Jan          |
|  2 |              |
|  3 | <podaj imie> |
|  4 | <podaj imie> |
+----+--------------+
4 rows in set (0.00 sec)

mysql>
```

Możesz oczywiście podrać tylko jedną kolumnę, na przykład tę zawierającą numery id:

```
mysql> SELECT id FROM ksiazka_adresowa;
+----+
| id |
+----+
|  1 |
|  2 |
|  3 |
|  4 |
+----+
4 rows in set (0.06 sec)

mysql>
```

A może chciałbyś wybrać maksymalną wartość, jaka znajduje się w kolumnie id? Do tego celu przydatna jest funkcja MAX():

```
mysql> SELECT MAX(id) FROM ksiazka_adresowa;
+---------+
| MAX(id) |
+---------+
|       4 |
+---------+
1 row in set (0.00 sec)

mysql>
```

Minimalną wartość pobiera się analogicznie, tylko za pomocą funkcji MIN():

```
mysql> SELECT MIN(id) FROM ksiazka_adresowa;
+---------+
| MIN(id) |
+---------+
|       1 |
+---------+
1 row in set (0.00 sec)

mysql>
```

Możesz również pobrać łączną liczbę rekordów za pomocą funcki COUNT():

```
mysql> SELECT COUNT(*) FROM ksiazka_adresowa;
+----------+
| COUNT(*) |
+----------+
|        4 |
+----------+
1 row in set (0.00 sec)

mysql>
```

Sumę wartości pól podanej kolumny można pobrać za pomocą funkcji SUM():

```
mysql> SELECT SUM(id) FROM ksiazka_adresowa;
+---------+
| SUM(id) |
+---------+
|      10 |
+---------+
1 row in set (0.00 sec)

mysql>
```

Średnią z wartości pól podanej kolumny pobiera się za pomocą funkcji AVG():

```
mysql> SELECT AVG(id) FROM ksiazka_adresowa;
+---------+
| AVG(id) |
+---------+
|  2.5000 |
+---------+
1 row in set (0.00 sec)

mysql>
```

Warunki:

Warunki w zapytaniu SELECT umożliwiają dokładniejsze określanie, jakie rekordy mają zostać wybrane. Możesz wybrać na przykład rekord o numerze id równym 3. Jak zwykle w warunkach możesz używać operatorów porównania. Poniższa tabela przedstawia dostępne operatory dla warunków:

=
Równość

<>, !=
Nierówność

<
Mniejsze

>
Większe

<=
Mniejsze lub równe

>=
Większe lub równe

Spójrzmy ma przykład wybierający rekord o numerze id równym 3:

```
mysql> SELECT * FROM ksiazka_adresowa WHERE id = 3;
+----+--------------+-------------------+-----------+-------+
| id | imie         | adres             | telefon   | email |
+----+--------------+-------------------+-----------+-------+
|  3 | <podaj imie> | ul. Warszawska 15 | 468-34-75 | NULL  |
+----+--------------+-------------------+-----------+-------+
1 row in set (0.06 sec)

mysql>
```

Możesz oczywiście podawać więcej niż jeden warunek. Do łączenia warunków służą operatory logiczne, takie jak AND i OR:

```
mysql> SELECT id, imie, telefon FROM ksiazka_adresowa WHERE (id >= 2 AND id < 4) OR (imie = 'Jan');
+----+--------------+-----------+
| id | imie         | telefon   |
+----+--------------+-----------+
|  1 | Jan          | 888-55-41 |
|  2 |              | 777-33-84 |
|  3 | <podaj_imie> | 468-34-75 |
+----+--------------+-----------+
3 rows in set (0.05 sec)

mysql>
```

Jak widać na powyższym przykładzie, grupy warunków umieszcza się w nawiasach.

Wyrażenie IN:

Zamiast podawać grupy warunków dla wartości, które mogą znaleźć się w grupie wyników, można posłużyć się wyrażeniem IN. Spójrzmy najpierw na przykład wybierania rekordów zawierających numer id równy 1, 2 lub 4:

```
mysql> SELECT id, adres FROM ksiazka_adresowa WHERE id = 1 OR id = 2 OR id = 4;
+----+------------------+
| id | adres            |
+----+------------------+
|  1 | ul. Wrocławska 5 |
|  2 | ul Krakowska 16  |
|  4 | NULL             |
+----+------------------+
3 rows in set (0.00 sec)

mysql>
```

To samo można zapisać znacznie prościej za pomocą wyrażenia IN, które definiuje grupę dopuszczalnych wartości dla pola:

```
mysql> SELECT id, adres FROM ksiazka_adresowa WHERE id IN(1, 2, 4);
+----+------------------+
| id | adres            |
+----+------------------+
|  1 | ul. Wrocławska 5 |
|  2 | ul Krakowska 16  |
|  4 | NULL             |
+----+------------------+
3 rows in set (0.00 sec)

mysql>
```

Jak widać użycie tego wyrażenia znacznie upraszcza zapytanie SQL.

Wyrażenie BEWEEN

Wyrażenie between (ang: pomiędzy) umożliwia określenie górnego i dolnego zakresu dla wartości pól rekordu. Spójrzmy najpierw na przykład wybierania rekordów o numerach id od 1 do 3 za pomocą standardowego WHERE:

```
mysql> SELECT id, telefon, email FROM ksiazka_adresowa WHERE id >= 1 AND id <= 3;
+----+-----------+--------------+
| id | telefon   | email        |
+----+-----------+--------------+
|  1 | 888-55-41 | jan@email.pl |
|  2 | 777-33-84 | zebra@zoo.pl |
|  3 | 468-34-75 | NULL         |
```

```
+----+-----------+--------------+
3 rows in set (0.00 sec)

mysql>
```

A teraz za pomocą wyrażenia BETWEEN:

```
mysql> SELECT id, telefon, email FROM ksiazka_adresowa WHERE id BETWEEN 1 AND 3;
+----+-----------+--------------+
| id | telefon   | email        |
+----+-----------+--------------+
|  1 | 888-55-41 | jan@email.pl |
|  2 | 777-33-84 | zebra@zoo.pl |
|  3 | 468-34-75 | NULL         |
+----+-----------+--------------+
3 rows in set (0.00 sec)

mysql>
```

Jak widać, użycie tego wyrażenia znacznie upraszcza zapytanie.

Wyrażenie NOT

Wyrażenie NOT pozwala zanegować wyrażenie takie jak IN czy BETWEEN. Spójrzmy na przykład wybierania rekordów z pominięciem rekordów o id równym 1 i 3:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa WHERE id NOT IN(1, 3);
+----+--------------+-----------------+
| id | imie         | adres           |
+----+--------------+-----------------+
|  2 |              | ul Krakowska 16 |
|  4 | <podaj_imie> | NULL            |
+----+--------------+-----------------+
2 rows in set (0.00 sec)

mysql>
```

Spójrzmy jeszcze na przykład wybierania rekordów, których numer id nie zawiera się w przedziale od 2 do 4:

```
mysql> SELECT id, telefon FROM ksiazka_adresowa WHERE id NOT BETWEEN 2 AND 4;
+----+-----------+
| id | telefon   |
+----+-----------+
|  1 | 888-55-41 |
+----+-----------+
1 row in set (0.00 sec)

mysql>
```

Sortowanie:

Domyślnie wybrane rekordy są sortowane w takiej kolejności, w jakiej zostały dodane do bazy. Można jednak tą kolejność zmienić używając polecenia ORDER BY.

Spójrzmy na przykład wybierający z bazy danych rekordy posortowane według imienia:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa ORDER BY imie;
+----+--------------+-------------------+
| id | imie         | adres             |
+----+--------------+-------------------+
|  2 |              | ul Krakowska 16   |
|  3 | <podaj_imie> | ul. Warszawska 15 |
|  4 | <podaj_imie> | NULL              |
|  1 | Jan          | ul. Wrocławska 5  |
+----+--------------+-------------------+
4 rows in set (0.00 sec)

mysql>
```

Kolejność rekordów zmieniła się, co najlepiej uwidacznia kolumna z numerami id. Spróbujmy jeszcze posortować rekordy według dwu pól: imienia i adresu:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa ORDER BY imie, adres;
+----+--------------+-------------------+
| id | imie         | adres             |
+----+--------------+-------------------+
|  2 |              | ul Krakowska 16   |
|  4 | <podaj_imie> | NULL              |
|  3 | <podaj_imie> | ul. Warszawska 15 |
|  1 | Jan          | ul. Wrocławska 5  |
+----+--------------+-------------------+
4 rows in set (0.00 sec)

mysql>
```

Jak widać, wyniki zostały posortowane według imienia, a te, które miały taką samą wartość w tym polu, zostały dodatkowo posortowane według pola adres.

Możesz wpływać na kierunek sortowania za pomocą słów kluczowych ASC (domyślnie) i DESC (kierunek odwrotny). Spróbujmy zatem posortować numery id w odwrotnej kolejności:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa ORDER BY id DESC;
+----+--------------+-------------------+
| id | imie         | adres             |
+----+--------------+-------------------+
|  4 | <podaj_imie> | NULL              |
|  3 | <podaj_imie> | ul. Warszawska 15 |
|  2 |              | ul Krakowska 16   |
```

```
|  1 | Jan          | ul. Wrocławska 5   |
+----+--------------+--------------------+
4 rows in set (0.00 sec)

mysql>
```

Słowo kluczowe DISTINCT

Czasem zachodzi potrzeba, aby wśród podanych w jednej kolumnie wyników zapytania nie powtarzały się wartości. Do tego celu służy słowo kluczowe DISTINCT.

Spójrzmy najpierw na normalne wybieranie wartości kolumny imie:

```
mysql> SELECT imie FROM ksiazka_adresowa;
+--------------+
| imie         |
+--------------+
| Jan          |
|              |
| <podaj_imie> |
| <podaj_imie> |
+--------------+
4 rows in set (0.00 sec)

mysql>
```

Jak widać wartość '<podaj imie>' występuje dwa razy. Spójrzmy teraz na wynik po użyciu słowa kluczowego DISTINCT:

```
mysql> SELECT DISTINCT imie FROM ksiazka_adresowa;
+--------------+
| imie         |
+--------------+
| Jan          |
|              |
| <podaj_imie> |
+--------------+
3 rows in set (0.00 sec)

mysql>
```

Jak widać wartości się już nie powtarzają.

Porównanie LIKE

Porównanie LIKE umożliwia tworzenie wzorców w zapytaniach. Spójrzmy na przykład wybierający pozycję w których pole imie na wartość '<podaj_imie>', za pomocą typowego warunku WHERE:

```
mysql> SELECT id, imie FROM ksiazka_adresowa WHERE imie = '<podaj_imie>';
+----+--------------+
| id | imie         |
+----+--------------+
|  3 | <podaj_imie> |
|  4 | <podaj_imie> |
+----+--------------+
2 rows in set (0.00 sec)

mysql>
```

A teraz za pomocą porównania LIKE:

```
mysql> SELECT id, imie FROM ksiazka_adresowa WHERE imie LIKE '<%>';
+----+--------------+
| id | imie         |
+----+--------------+
|  3 | <podaj_imie> |
|  4 | <podaj_imie> |
+----+--------------+
2 rows in set (0.00 sec)

mysql>
```

Jak widać, wynik jest ten sam, a możliwości większe. Znak % zonacza dowolny ciąg znaków, natomiast znak _ oznacza dokładnie jeden znak:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa WHERE imie LIKE '_an';
+----+------+------------------+
| id | imie | adres            |
+----+------+------------------+
|  1 | Jan  | ul. Wrocławska 5 |
+----+------+------------------+
1 row in set (0.00 sec)

mysql>
```

Porcjowanie wyników zapytania:

Jeżeli nie interesują cię wszystkie rekordy pasujące do zapytania, a na przykład jedynie pierwsze 10, możesz posłużyć się klauzulą LIMIT.

Spójrzmy na przykład wybierający z naszej książki adresowej tylko pierwsze dwa rekordy:

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa LIMIT 2;
+----+------+------------------+
| id | imie | adres            |
+----+------+------------------+
|  1 | Jan  | ul. Wrocławska 5 |
|  2 |      | ul Krakowska 16  |
```

```
+----+------+------------------+
2 rows in set (0.00 sec)

mysql>
```

Jak widać, aby zaznaczyć, że chcesz otrzymać określoną liczbę rekordów, powinieneś posłużyć się klauzulą LIMIT, po której podasz odpowiednią wartość.

Spójrzmy na przykład pobierania dwóch rekordów, począwszy od drugiego (dla MySQL-a będzie to jednak rekord 1, ponieważ tutaj rekordy liczy się, zaczynając od zera):

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa LIMIT 1, 2;
+----+--------------+------------------+
| id | imie         | adres            |
+----+--------------+------------------+
|  2 |              | ul Krakowska 16  |
|  3 | <podaj_imie> | ul. Warszawska 15 |
+----+--------------+------------------+
2 rows in set (0.00 sec)

mysql>
```

Jak widać z powyższego przykładu, aby pobrać określoną liczbę rekordów, począwszy od ustalonego, podajemy po przecinku numer rekordu początkowego i liczbę rekordów z zbiorze zapytania.

Pierwszy przykład pobierania dwóch rekordów można również zapisać, podając rekord, od którego ma się zacząć pobieranie wyników (czyli podamy zera jako pozycję pierwszego rekordu):

```
mysql> SELECT id, imie, adres FROM ksiazka_adresowa LIMIT 0, 2;
+----+------+------------------+
| id | imie | adres            |
+----+------+------------------+
|  1 | Jan  | ul. Wrocławska 5 |
|  2 |      | ul Krakowska 16  |
+----+------+------------------+
2 rows in set (0.00 sec)

mysql>
```

Modyfikacja rekordów

Zauważyłeś pewnie że pozycja dodana jako druga, po skrócie 'ul' nie zawiera kropki. Spróbujmy teraz to poprawić za pomocą polecenia UPDATE. Składnia tego polecenia przedstawia się następująco:

```
UPDATE <nazwa_bazy>.<nazwa_tabeli> SET <nazwa_pola> = 'wartość' WHERE <warunek>

lub

UPDATE <nazwa_tabeli> SET <nazwa_pola> = 'wartość' WHERE <warunek>
```

jeżeli wcześniej zostało użyte polecenie use <nazwa_bazy>.

Spróbujmy zatem poprawić wartość pola adres, które ma id numer 2:

```
mysql> UPDATE ksiazka_adresowa SET adres = 'ul. Krakowska 16' WHERE id = 2;
Query OK, 1 row affected (0.05 sec)
Rows matched: 1 Changed: 1 Warnings: 0

mysql>
```

Spójrzmy teraz na rekord o numerze id równym 2:

```
mysql> SELECT * FROM ksiazka_adresowa WHERE id = 2;
+----+------+------------------+-----------+--------------+
| id | imie | adres            | telefon   | email        |
+----+------+------------------+-----------+--------------+
|  2 |      | ul. Krakowska 16 | 777-33-84 | zebra@zoo.pl |
+----+------+------------------+-----------+--------------+
1 row in set (0.00 sec)

mysql>
```

Usuwanie rekordów

Na koniec zajmiemy się usuwaniem rekordów, Służy do tego celu polecenie DELETE, którego składnia przedstawia się następująco:

```
DELETE FROM <nazwa_bazy>.<nazwa_tabeli> WHERE <warunek>

lub

DELETE FROM <nazwa_tabeli> WHERE <warunek>
```

jeżeli wcześniej zostało użyte polecenie use <nazwa_bazy>.

Jeżeli warunek WHERE nie zostanie podany, zawartość całej tabeli zostanie usunięta.

Spróbujmy teraz usunąć rekord o id równym 3:

```
mysql> DELETE FROM ksiazka_adresowa WHERE id = 3;
Query OK, 1 row affected (0.06 sec)

mysql>
```

I spójrzmy jak teraz wygląda zawartość naszej tabeli 'ksiazka_adresowa':

```
mysql> SELECT * FROM ksiazka_adresowa;
+----+--------------+------------------+-----------+--------------+
| id | imie         | adres            | telefon   | email        |
+----+--------------+------------------+-----------+--------------+
|  1 | Jan          | ul. Wrocławska 5 | 888-55-41 | jan@email.pl |
```

```
|  2 |              | ul. Krakowska 16 | 777-33-84 | zebra@zoo.pl |
|  4 | <podaj_imie> | NULL             | 471-84-36 | my@email.pl  |
+----+--------------+------------------+-----------+--------------+
3 rows in set (0.00 sec)

mysql>
```

Jak widać, wartości numerów id nie przeindeksowały się (pole auto_increment), a następny numer id, jaki zostanie dodany, będzie miał wartość o jeden większą od największego, czyli 5.

Użytkownicy i uprawnienia
Ta część kursy przybliży ci temat określania uprawnień dla użytkowników bazy danych.
System uprawnień w MySQL-u pozwala na dokładne zdefiniowanie praw dostępu do określonej bazy, tabeli czy kolumny.
System uprawnień MySQL-a opiera się na pięciu tabelach znajdujących się w bazie mysql. Poniższy przykład prezentuje ich nazwy:

```
mysql> use mysql
Database changed

mysql> show tables;
+-----------------+
| Tables_in_mysql |
+-----------------+
| columns_priv    |
| db              |
| host            |
| tables_priv     |
| user            |
+-----------------+
5 rows in set (0.05 sec)

mysql>
```

Każda z tych tabel określa uprawnienia użytkownika na innym poziomie dostępu. Tablela db określa dostęp do bazy danych, host definiuje dostęp danych dostów do baz danych, user określa użytkowników całego serwera bazy danych, a tabele tables_priv i columns_priv określają uprawnienia odpowiednio do tabel i do ich kolumn.

Tabela USER

Przyjrzyjmy się teraz bliżej tabeli user, która definiuje użytkowników i ich uprawnienia w naszym systemie. Najpierw spójrzmy na strukturę tej tabeli:

```
mysql> describe user;
+-----------------+---------------+------+-----+-----+-------+-------+
| Field           | Type          | Null | Key | ()* | Extra | ()**  |
+-----------------+---------------+------+-----+-----+-------+-------+
| Host            | char(60)      |      | PRI |     |       | ()*** |
| User            | char(16)      |      | PRI |     |       | ()*** |
| Password        | char(16)      |      |     |     |       | ()*** |
| Select_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Insert_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Update_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Delete_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Create_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Drop_priv       | enum('N','Y') |      |     | N   |       | ()*** |
| Reload_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Shutdown_priv   | enum('N','Y') |      |     | N   |       | ()*** |
| Process_priv    | enum('N','Y') |      |     | N   |       | ()*** |
| File_priv       | enum('N','Y') |      |     | N   |       | ()*** |
| Grant_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| References_priv | enum('N','Y') |      |     | N   |       | ()*** |
| Index_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| Alter_priv      | enum('N','Y') |      |     | N   |       | ()*** |
+-----------------+---------------+------+-----+-----+-------+-------+
17 rows in set (0.11 sec)

mysql>
```

Objaśnienia:
  * Default
 ** Privileges
*** select,insert,update,references

Jak widać ta tabela zawiera pola dla nazwy hosta i użytkownika, hasła oraz listę pól z wartościami N lub Y dla określonych praw.

Spójrzmy co zawiera ta tabela:

```
mysql> select * from user;
+-----------+------+----------+-------------+-------------+-------------+-------------+-------------+-------------+---------------+--------------+
+-----------+------------+-----------------+-----------+-----------+
| Host      | User | Password | Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv | Drop_priv | Reload_priv | Shutdown_priv | Process_priv |
File_priv | Grant_priv | References_priv | Index_priv | Alter_priv |
+-----------+------+----------+-------------+-------------+-------------+-------------+-------------+-------------+---------------+--------------+
+-----------+------------+-----------------+-----------+-----------+
| localhost | root |          | Y           | Y           | Y           | Y           | Y           | Y           | Y             | Y            | Y
| Y         | Y          | Y               | Y         |           |
| %         |      |          | N           | N           | N           | N           | N           | N           | N             | N            | N
| N         | N          | N               | N         |           |
| localhost |      |          | Y           | Y           | Y           | Y           | Y           | Y           | Y             | Y            | Y
| Y         | Y          | Y               | Y         |           |
| %         | root |          | Y           | Y           | Y           | Y           | Y           | Y           | Y             | Y            | Y
| Y         | Y          | Y               | Y         |           |
+-----------+------+----------+-------------+-------------+-------------+-------------+-------------+-------------+---------------+--------------+
+-----------+------------+-----------------+-----------+-----------+
4 rows in set (0.11 sec)

mysql>
```

Jak widać, tabela user zawiera cztery wpisy:

użytkownik root może zalogować się z localhost, nie podaje hasła, ma uprawnienia do każdej operacji. To właśnie jako ten użytkownik cały czas pracujemy, logując się komendą mysql -u root,

użytkownik bez zdefiniowanej nazwy, z dowolnego hosta (znak '%' oznacza dowolny adres hosta), nie ma żadnych uprawnień, nie podaje

hasła,

użytkownik bez zdefiniowanej nazwy, z localhost, ma podobnie jak root wszelkie uprawnienia, nie podaje hasła,

użytkownik root może zalogować się z dowolnego hosta, nie podaje hasła, ma uprawnienia do każdej operacji.

Tabela DB

Przyjrzyjmy się teraz bliżej tabeli db, która określa dostęp do bazy danych. Najpierw spójrzmy na strukturę tej tabeli:

```
mysql> describe db;
+-----------------+---------------+------+-----+-----+-------+-------+
| Field           | Type          | Null | Key | ()* | Extra | ()**  |
+-----------------+---------------+------+-----+-----+-------+-------+
| Host            | char(60)      |      | PRI |     |       | ()*** |
| Db              | char(64)      |      | PRI |     |       | ()*** |
| User            | char(16)      |      | PRI |     |       | ()*** |
| Select_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Insert_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Update_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Delete_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Create_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Drop_priv       | enum('N','Y') |      |     | N   |       | ()*** |
| Grant_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| References_priv | enum('N','Y') |      |     | N   |       | ()*** |
| Index_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| Alter_priv      | enum('N','Y') |      |     | N   |       | ()*** |
+-----------------+---------------+------+-----+-----+-------+-------+
13 rows in set (0.11 sec)

mysql>
```

Objaśnienia:
  * Default
 ** Privileges
*** select,insert,update,references

Jak widać, tabela ta zawiera pola dla nazwy hosta, bazy danych i użytkownika oraz listę pól z wartościami N lub Y dla określonych praw.

Spójrzmy co już zawiera ta tabela:

```
mysql> select * from user;
+------+------+------+-------------+-------------+-------------+-------------+-------------+-----------+------------+-----------------+------------
+------------+
| Host | Db    | User | Select_priv | Insert_priv | Update_priv | Delete_priv | Create_priv | Drop_priv | Grant_priv | References_priv | Index_priv | Alter_priv
|
+------+------+------+-------------+-------------+-------------+-------------+-------------+-----------+------------+-----------------+------------
+------------+
| %    | test% |      | Y           | Y           | Y           | Y           | Y           | Y         | N          | Y               | Y          | Y
|
+------+------+------+-------------+-------------+-------------+-------------+-------------+-----------+------------+-----------------+------------
+------------+
1 row in set (0.00 sec)
```

Tabela HOST

Przyjrzyjmy się teraz bliżej tabeli host, która dokładniej definiuje przywileje zawarte w tabeli db. Najpierw spójrzmy na strukturę tej tabeli:

```
mysql> describe host;
+-----------------+---------------+------+-----+-----+-------+-------+
| Field           | Type          | Null | Key | ()* | Extra | ()**  |
+-----------------+---------------+------+-----+-----+-------+-------+
| Host            | char(60)      |      | PRI |     |       | ()*** |
| Db              | char(64)      |      | PRI |     |       | ()*** |
| Select_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Insert_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Update_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Delete_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Create_priv     | enum('N','Y') |      |     | N   |       | ()*** |
| Drop_priv       | enum('N','Y') |      |     | N   |       | ()*** |
| Grant_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| References_priv | enum('N','Y') |      |     | N   |       | ()*** |
| Index_priv      | enum('N','Y') |      |     | N   |       | ()*** |
| Alter_priv      | enum('N','Y') |      |     | N   |       | ()*** |
+-----------------+---------------+------+-----+-----+-------+-------+
12 rows in set (0.05 sec)

mysql>
```

Objaśnienia:
  * Default
 ** Privileges
*** select,insert,update,references

Jak widać, tabela ta zawiera pola dla nazwy hosta i bazy danych oraz listę pól z wartościami N i Y dla określonych praw.

Spójrzmy co już zawiera ta tabela:

```
mysql> select * from host;
Empty set (0.00 sec)
```

Jak widać, tabela host nie zawiera żadnych wpisów, ponieważ nie były jeszcze potrzebne tak dokładnie określone uprawnienia dostępu.

Tabela TABLES_PRIV

Przyjrzyjmy się teraz bliżej tabeli tables_priv, która definiuje przywileje dostępu do określonych tabel, podobnie jak tabela db do baz danych. Najpierw spójrzmy na jej strukturę:

```
mysql> describe tables_priv;
+-------------+---------------+------+-----+------+-------+-------+
```

```
| Field       | Type          | Null | Key | ()*  | Extra | ()**  |
+-------------+---------------+------+-----+------+-------+-------+
| Host        | char(60)      |      | PRI |      |       | ()*** |
| Db          | char(64)      |      | PRI |      |       | ()*** |
| User        | char(16)      |      | PRI |      |       | ()*** |
| Table_name  | char(64)      |      | PRI |      |       | ()*** |
| Grantor     | char(77)      |      |     |      |       | ()*** |
| Timestamp   | timestamp(14) | YES  |     | NULL |       | ()*** |
| Table_priv  | Patrz 1)      |      |     |      |       | ()*** |
| Column_priv | Patrz 2)      |      |     |      |       | ()*** |
+-------------+---------------+------+-----+------+-------+-------+
8 rows in set (0.08 sec)

mysql>
```

Objaśnienia:
  * Default
 ** Privileges
*** select,insert,update,references

```
1) set('Select','Insert','Update','Delete','Create',
       'Drop','Grant','References','Index','Alter')

2) set('Select','Insert','Update','References')
```

Jak widać, tabela ta zawiera pola dla nazwy hosta, bazy danych, użytkownika, tabeli i użytkownika nadającego przywilej oraz pola przeznaczone do określania uprawnień dostępu.

Spójrzmy co zawiera ta tabela:

```
mysql> select * from tables_priv;
Empty set (0.00 sec)

mysql>
```

Jak widać, tabela host nie zawiera żadnych wpisów, ponieważ nie były potrzebne tak dokładnie określone uprawnienia dostępu.

Tabela COLUMNS_PRIV

Przyjrzyjmy się teraz bliżej tabeli columns_priv, która definiuje przywileje dostępu do określonych kolumn w tabelach. Najpierw spójrzmy na strukturę tej tabeli:

```
mysql> describe columns_priv;
+-------------+---------------+------+-----+------+-------+-------+
| Field       | Type          | Null | Key | ()*  | Extra | ()**  |
+-------------+---------------+------+-----+------+-------+-------+
| Host        | char(60)      |      | PRI |      |       | ()*** |
| Db          | char(64)      |      | PRI |      |       | ()*** |
| User        | char(16)      |      | PRI |      |       | ()*** |
| Table_name  | char(64)      |      | PRI |      |       | ()*** |
| Column_name | char(64)      |      | PRI |      |       | ()*** |
| Timestamp   | timestamp(14) | YES  |     | NULL |       | ()*** |
| Column_priv | Patrz 1)      |      |     |      |       | ()*** |
+-------------+---------------+------+-----+------+-------+-------+
7 rows in set (0.00 sec)

mysql>
```

Objaśnienia:
  * Default
 ** Privileges
*** select,insert,update,references

```
1) set('Select','Insert','Update','References')
```

Jak widać, tabela ta zawiera pola dla nazwy hosta, bazy danych, użytkownika, tabeli i nazwy kolumny oraz pole przeznaczone do określania uprawnień dostępu.

Spójrzmy co już zawiera ta tabela:

```
mysql> select * from columns_priv;
Empty set (0.00 sec)

mysql>
```

Jak widać, tabela columns_priv nie zawiera żadnych wpisów, ponieważ nie były jeszcze potrzebne tak dokładnie określone uprawnienia dostępu.

Manipulacja uprawnieniami

Po dłuższym wstępie przejdźmy do nieco ciekawszej części, czyli nadawania uprawnień i ich odbierania. Wszelkie zmiany dotyczące tabel w bazie mysql możesz przeprowadzać za pomocą znanych ci z poprzedniej części kursy poleceń, takich jak select, insert, update czy delete. Możesz również w łatwy sposób zarządzać uprawnieniami za pomocą poleceń GRANT (dodawanie uprawnień) i REVOKE (odbieranie uprawnień).
Na początku ustawmy hasło dla użytkownika root, aby nikt niepowołany nie mógł w łatwy sposób dostać się do serwera baz danych:

```
mysql> UPDATE user SET Password = PASSWORD('pass') WHERE user = 'root';
Query OK, 2 rows affected (0.05 sec)
Rows matched: 2 Changed: 2 Warnings: 0

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Za pomocą polecenia UPDATE zmieniliśmy wartość pola Password na nowe hasło, tam gdzie użytkownik jest zdefiniowany jako root (dwa rekordy). Hasła w MySQL kodowane są za pomocą funkcji PASSWORD(), dlatego musisz jej użyć, aby zapisać hasło w tabeli.

Ostatnie polecenie (FLUSH PRIVILEGES;) przeładowuje uprawnienia. Gdybyśmy tego nie zrobili, zmiany nie byłyby widoczne, root dalej logowałby się bez hasła. Nie musisz odświeżać uprawnień, jeżeli używasz wbudowanych funkcji do obsługi przywilejów:

```
mysql> SET PASSWORD FOR root = PASSWORD('mypass');
Query OK, 0 rows affected (0.05 sec)

mysql>
```

Spróbujmy teraz zalogować się do mysql-a z hasłem 'pass':

```
C:\apache\mysql\bin> mysql -u root -p
Enter password: ****
ERROR 1045: Access denied for user: 'root@localhost' (using password: YES)

C:\apache\mysq\bin>
```

Jak widać hasło 'pass' nie jest już aktualne, ponieważ zmieniliśmy je na 'mypass' za pomocą funkcji SET PASSWORD...

Spróbujmy zatem jeszcze raz, tym razem podając haslo 'mypass':

```
C:\apache\mysql\bin> mysql -u root -p
Enter password: ******

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 40 to server version: 3.23.33
Type 'help' for help

mysql>
```

I spróbujmy jeszcze zalogować się bez hasła, jak dawniej:

```
C:\apache\mysql\bin> mysql -u root
ERROR 1045: Access denied for user 'root@localhost' (Using password: NO)

C:\apache\mysql\bin>
```

Jak wiadć, nasz MySQL jest zabezpieczony hasłem.

Polecenie GRANT

Za pomocą tego polecenia dodajesz uprawnienia dostępu do baz danych, tabel czy kolumn.

Spróbujmy teraz utworzyć nowego użytkownika admin z dostępem do bazy danych nasza_baza i tabeli ksiazka_adresowa:

```
mysql> GRANT SELECT ON nasza_baza.ksiazka_adresowa TO admin@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Mamy już w tej chwili dostępnego użytkownika admin, który ma prawo wybierać dane z tabeli ksiazka_adresowa z naszej bazy. Ten użytkownik nie ma ustawionego hasła, ale musi logować się z localhost:

```
C:\apache\mysql\bin> mysql -u admin

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 52 to server version: 3.23.33
Type 'help' for help.

mysql>
```

Jak widać, nowy użytkownik może się już zalogować. Przetestujmy teraz jego uprawnienia, najpierw na bazie mysql, potem na nasza_baza:

```
mysql> use mysql
ERROR 1044: Access denied for user: 'admin@localhost' to database 'mysql'

mysql> use nasza_baza
Database changed

mysql>INSERT INTO ksiazka_adresowa VALUES(0, 'Imie', 'adres', 'telefon', 'adres@email.pl');
ERROR 1142: insert command denied to user 'admin@localhost' for table 'ksiazka_adresowa'

mysql> select * from ksiazka_adresowa;
Empty set (0.00 sec)

mysql>
```

Jak wynika z powyższego przykładu, nasz nowy użytkownik nie ma dostępu do bazy mysql. Nie może również dodawać rekordów do tabeli ksiazka_adresowa w naszej bazie. Jedynie polecenie SELECT zostało zakończone sukcesem, bo tylko do tego ma uprawnienia użytkownik admin.

Spróbujmy utworzyć teraz użytkownika admin2 z uprawnieniami do wybierania, dodawania, edycji i usuwania rekordów:

```
mysql> GRANT SELECT,INSERT,UPDATE,DELETE ON nasza_baza.* TO admin2@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Ten użtykownik ma prawo do wykonywania instrukcji SELECT, INSERT, UPDATE i DELETE na wszystkich tabelach w bazie nasza_baza. Znak * użyty zamiast nazwy tabeli oznacza dostęp do wszystkich tabel.

Spróbujmy teraz utworzyć użytkownika admin3 z prawem dodawania rekordów we wszystkich bazach danych systemu:

```
mysql> GRANT INSERT ON *.* TO admin3@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Jak widać, aby zaznaczyć wszystkie bazy danych wraz z towarzyszącymi im tabelami, użyliśmy konstrukcji *.*, co oznacza <dowolna_baza>.<dowolna_tabela>:.

Spróbujmy teraz utworzyć użytkownika admin4 przychodzącego z dowolnego hosta z prawem wybierania rekordów z bazy nasza_baza i dowolnej tabeli:

```
mysql> GRANT SELECT ON nasza baza.* TO admin4@'%';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Jak widać, aby oznaczyć dowolny host, użyliśmy znaku % umieszczonego w cudzysłowach.

Spróbujmy teraz utworzyć użytkownika admin5 ze wszystkimi prawami w bazie nasza_baza i dowolnej tabeli przychodzącego z localhost:

```
mysql> GRANT ALL PRIVILEGES ON nasza_baza.* TO admin5@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Jak widać, aby zaznaczyć, że chcemy nadać użytkownikowi wszystkie prawa, użyliśmy polecenia ALL PRIVILEGES.

Spróbujmy jeszcze utworzyć takiego użytkownika jak poprzedni, ale z prawem nadawania uprawnień (grantów):

```
mysql> GRANT ALL PRIVILEGES ON nasza_baza.* TO admin6@localhost WITH GRANT OPTION;
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Jak widać, aby zaznaczyć, że chchcemy nadać wszystkie prawa razem z grantami, użyliśmy polecenia WITH GRANT OPTION.

Spróbujmy teraz utworzyć użytkownika admin7 z dostępem do wszystkich baz danych i tabel, ale z wymaganym hasłem 'haslo':

```
mysql> GRANT ALL PRIVILEGES ON *.* TO admin7@localhost IDENTIFIED BY 'haslo';
Query OK, 0 rows affected (0.00 sec)

mysql>
```

Spróbujmy teraz zalogować się na użytkownika admin7, raz bez podania hasła, a następnie z hasłem:

```
C:\apache\mysql\bin> mysql -u admin7
ERROR 1045: Access denied for user: 'admin7@localhost' (Using password: NO)

C:\apache\mysq\bin> mysql -u admin7 -p
Enter password: *****

Welcome to the MySQL monitor. Commands with ; or \g.
Your MySQL connection id is 78 to server version: 3.23.33
Type 'help' for help.

mysql>
```

Jak widać hasło dla tego użytkownika jest wymagane.

Spróbujmy jeszcze utworzyć użytkownika admin8 z prawem wybierania pola imie oraz edycji pól adres i email z tabeli ksiazka_adresowa w naszej baze testowej:

```
mysql> GRANT SELECT(imie),UPDATE(adres,email) ON nasza_baza.ksiazka_adresowa TO admin8@localhost;
Query OK, 0 rows affected (0.22 sec)

mysql>
```

Jak widać, aby określić kolumny, na jakich mają działać uprawnienia, wystarczy zapisać je w nawiasach okrągłych występujących po uprawnieniu.

Polecenie REVOKE

Za pomocą tego polecenia usuwasz uprawnienia dostępu do baz danych, tabel czy kolumn.

Spróbujmy teraz usunąć uprawnienia nadane użytkownikowi admin z dostępem do bazy danych nasza_baza i tabeli ksiazka_adresowa:

```
mysql> REVOKE SELECT ON nasza_baza.ksiazka_adresowa FROM admin@localhost;
Query OK, 0 rows affected (0.05 sec)

mysql>
```

Jak widać z powyższego przykładu, aby usunąć uprawnienia nadane użytkownikowi, posługujemy się podobną składnią jak przy dodawaniu uprawnień. Różnice są dwie: zamiast słowa GRANT występuje REVOKE, a zamiast TO - FROM.

Spróbujmy sprawdzić, czy rzeczywiście nie mamy dostępu do bazy danych nasza_baza:

```
C:\apache\mysql\bin> mysql -u admin

Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 83 to server version: 3.23.33
Type 'help' for help.

mysql> use nasza_baza
ERROR 1044: Access denied for user: 'admin@localhost' to database 'nasza_baza'

mysql>
```

Na koniec spróbujmy jeszcze odebrać uprawnienia użytkownikowi admin8:

```
mysql> REVOKE SELECT(imie),UPDATE(adres,imie) ON nasza_baza.ksiazka_adresowa FROM admin8@localhost;
Query OK, 0 rows affected (0.00 sec)

mysql>
```